Seminar

Software Engineering für verteilte Systeme

Sommersemester 2024

Recent Approaches to accelerate CP Solving

–

Matrikelnummer: –

––e

Betreuer: –

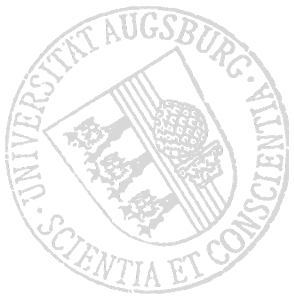
Softwaremethodik für verteilte Systeme (Prof. Bauer)

Universität Augsburg

Zusammenfassung Constraint Programming (CP) spielt eine entscheidende Rolle bei der Lösung von Optimierungsproblemen mit Nebenbedingungen in verschiedenen Anwendungen wie Fahrzeugroutenplanung, Zeitplanung und Konfiguration. Aktuelle Ansätze zielen darauf ab, CP-Systeme zu beschleunigen, um schnellere Lösungen zu ermöglichen.

Portfolio-Ansätze kombinieren mehrere Solver und wählen oder kombinieren sie dynamisch basierend auf den Problemmerkmalen aus. Die modellbasierte Optimierung verwendet Ersatzmodelle, während das automatische Parameter-Tuning Solverparameter automatisch anpasst. Die Kombination von CP mit SAT und Lazy Clause Generation zeigt vielversprechende Ergebnisse für die effiziente Lösung komplexer Probleme.

Die Bemühungen zur Beschleunigung von CP-Systemen bieten vielfältige Strategien, wobei deren Wirksamkeit stark von den spezifischen Problemstellungen abhängt. Die laufende Forschung verspricht weitere Verbesserungen, die möglicherweise von aufkommenden Technologien wie der Quantencomputing beeinflusst werden.


UNIVERSITÄT AUGSBURG · SCIENTIA ET CONSCIENTIA

Inhaltsverzeichnis

1 Einleitung: Verwendung von Constraint Programming

Constraint Programming (CP) spielt in vielen modernen Anwendungen eine große Rolle, in denen Optimierungsprobleme mit Nebenbedingungen gelöst werden müssen. Anwendungen hierzu sind unter anderem: Vehicle Routing, Scheduling, Planung, Konfiguration, Ressourcenallokation und Kombinatorische Optimierung. Jedes Jahr findet die Conference on Principles and Practice of Constraint Programming statt, in der aktuelle Forschungsergebnisse im Bereich CP diskutiert werden, was die Wichtigkeit des Themenfeldes unterstreicht [1]. Neben dem klassischen Problem der Kursplanung [2] in einer sich ständig beschleunigenden Zeit gibt es auch in der Industrie eine Vielzahl von Scheduling-Problemen, wie die Zuweisung von Aufträgen zu Maschinen [3]. Auch wurden CP-Ansätze für die Konfiguration von Netzwerken verwendet [4]. Ein weit verbreitetes kombinatorisches Optimierungsproblem ist das 3-SAT-Problem, das beschreibt, ob eine Formel in konjunktiver Normalform (KNF) mit Klauseln aus jeweils 3 Literalen erfüllbar ist [5, S. 271].

Beispielsweise löst

$$x_1 = \text{false}, x_2 = \text{false}, x_3 = \text{false}, x_4 = \text{false}$$

das folgende 3-SAT-Problem:

$$(\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_4) \wedge (x_2 \vee x_3 \vee \neg x_4)$$

Besonders Vehicle Routing ist in der heutigen Zeit für viele Unternehmen in der Logistikbranche und dem flexiblen Transport von großer Bedeutung [6, S. 1]. Je nach Anwendungen und Problemstellung können unterschiedliche Constraint Programming Ansätze verwendet werden. Es existieren verschiedene Toolsets, um Constraint Programming-Probleme zu lösen. Für das Vehicle Routing Problem, welches das Problem beschreibt, den kürzesten Gesamtweg für m Fahrzeuge und n Orte zu finden [7, S. 222], kann beispielsweise der IBM ILOG CP Optimizer verwendet werden [8]. Je nach Problemstellung gibt es eine Vielzahl weiterer Toolsets [9]. OR-tools von Google ist eine weitere Möglichkeit zur Lösung von Constraint Programming (CP), Linear Programming (LP), Integer Programming (IP) und Boolean Satisfiability (SAT) Problemen [10]. Geocode ist eine Open-Source-Variante basierend auf C++ [11], und Chuffed ist eine Variante, die “lazy clause generation“ ausnutzt, um CP-Probleme schneller zu lösen [12]. MiniZinc vereint unter anderem die zuvor beschriebenen Softwarebibliotheken zu einer Ausdruckssprache zum Lösen einer Vielzahl von LP, Transportproblemen (TP) und SAT-Problemen [13]. Unter demselben Namen wird jährlich die MiniZinc Challenge ausgetragen, in der ein Parcours von Constraint-Modellen gelöst werden muss. Am Ende werden die Solver nach Anzahl der gelösten Modelle, der Zeit und der Qualität der Lösung bewertet [14]. Gerade bei großen Problemstellungen, oder wenn, wie in der Forschung oft üblich, mehrere Lösungen benötigt werden oder besondere Ansprüche an die Qualität und Genauigkeit der Lösung gestellt werden, kann die Ausführung oft lange dauern, weshalb die benötigte Zeit der Solver von großer Bedeutung ist. Die folgende Arbeit soll aktuelle Ansätze vorstellen, mit denen CP-Probleme schneller gelöst werden können.

2 Grundlagen

Grundsätzlich gibt es zwei Arten von Constraint-Problemen: Constraint Satisfaction Probleme (CSP) und Constraint Optimization Probleme (COP). Ein CSP beschreibt im Wesentlichen ein Problem, bei dem Nebenbedingungen festlegen, welche Werte die Variablen annehmen können [5, S. 13]. Das Ziel hierbei kann es entweder sein zu überprüfen, ob eine Lösung existiert, eine mögliche Lösung zu finden oder die Menge aller möglichen Lösungen zu bestimmen. Ein COP ist eine Erweiterung des CSP, bei dem zusätzlich eine Zielfunktion minimiert werden muss [5, S. 171]. Ebenso kann es hierbei auch darum gehen zu überprüfen, ob das Problem überhaupt lösbar ist, eine mögliche Lösung zu finden oder die optimale Lösung zu finden.

2.1 Constraint Satisfaction Probleme

Ein CSP lässt sich durch ein Tripel $P = (X, D, C)$ beschreiben, wobei $X = \langle x_1, x_2, \ldots, x_n \rangle$ ein n-Tupel von Variablen, $D = \langle D_1, D_2, \ldots, D_n \rangle$ ein n-Tupel von Domänen ist, so dass $x_i \in D_i$ erfüllt ist und $C = \langle C_1, C_2, \ldots, C_t \rangle$ eine Menge von Bedingungen beschreibt. Jede Bedingung C_j ist hierbei eine Untermenge des kartesischen Produkts über D. Eine Lösung für ein CSP ist ein n-Tupel $A = \langle a_1, a_2, \ldots, a_n \rangle$, so dass $a_i \in D_i$ für alle i und $A \in C_j \quad \forall j$ [5, S. 16]. Ein Beispiel für ein CSP ist das oben beschriebene 3-SAT-Problem. Das Lösen eines 3-SAT-Problems ist NP-vollständig, was bedeutet, dass es keinen effizienten Algorithmus gibt, der das Problem in Polynomialzeit lösen kann [5, S. 17]. Eine einfache Möglichkeit, ein CSP zu lösen, ist die Verwendung von Backtracking. Hierbei wird eine Variable nach der anderen belegt, und bei einem Widerspruch wird ein Schritt zurückgegangen und eine andere Variable belegt [5, S. 85]. Interessant ist auch, dass sich auch alle $k > 3$-SAT-Probleme auf ein 3-SAT-Problem reduzieren lassen, wodurch auch das $k > 3$ SAT-Problem NP-vollständig ist [15, S. 206]. K-SAT Klauseln lassen sich nach folgendem Schema durch Einführung einer Hilfsvariable in K-1-SAT Klauseln herunterbrechen: $x_1 \vee x_2 \vee x_3 \vee x_4$ ist äquivalent zu $(x_1 \vee x_2 \vee y) \wedge (\neg y \vee x_3 \vee x_4)$ .Sind die Domänen der Variablen auf boolesche Werte beschränkt, handelt es sich um ein Boolean Satisfiability Problem (SAT).

2.2 Constraint Optimization Probleme

Viele Probleme in der realen Welt suchen nicht nur eine Lösung, sondern die optimale Lösung. Solche Probleme lassen sich durch ein Constraint Optimization Problem modellieren. Hierzu wird die Problemstellung um eine Zielfunktion erweitert, die minimiert oder maximiert werden soll. Ein COP lässt sich durch ein Tripel $P = (X, D, C, f)$ beschreiben, wobei die ersten drei Elemente wie bei einem CSP sind und f eine Zielfunktion ist [16, S. 22]. In der Physik oder den Ingenieurwissenschaften werden COP oft auch in Funktionsdarstellung beschrieben.

$$
\begin{aligned}
&\text{minimiere:} \quad f(x) \\
&\text{unter der Bedingung:} \\
&\quad g_j(x) \leq 0 \\
&\quad h_l(x) = 0 \\
&\quad \underline{x_i} \leq x_i \leq \overline{x_i}
\end{aligned}
$$

Hierbei beschreiben $g_j(x)$ und $h_l(x)$ die Nebenbedingungen C, wobei erstere eine Ungleichungsrestriktion und letztere eine Gleichungsrestriktion darstellt. Die letzte Gleichung beschreibt die Domäne D der Variablen [17, S. 154]. Je nach Art der Problemstellung unterscheiden sich die verwendeten Algorithmen. Sind beispielsweise die Nebenbedingungen und die Zielfunktion linear, handelt es sich um ein Linear Programming Problem (LP). Dieses lässt sich beispielsweise mit dem Simplex-Algorithmus lösen. Das Verfahren beruht darauf, dass die Nebenbedingungen einen n-dimensionalen Polyeder aufspannen und die optimale Lösung auf einer Ecke des Polyeders liegt. Der Simplex-Algorithmus sucht nun nacheinander die Ecken ab, bis die optimale Lösung gefunden wurde. Beschränkt man sich bei den Lösungen auf ganzzahlige Werte, spricht man von einem Integer Programming Problem (IP). Diese lassen sich mit dem Branch-and-Bound-Algorithmus lösen [18] [19, S. 99].

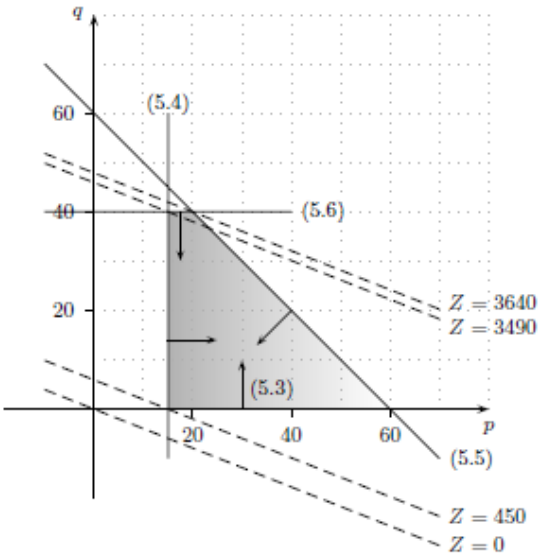


Abbildung 1: Der Polyeder beschreibt die Nebenbedingungen, gestrichelte Linie die Zielfunktion, Ecken sind mögliche Lösungen [19, S. 100]

Oft lassen sich Probleme jedoch nur über nichtlineare Zusammenhänge beschreiben. Eine Möglichkeit, solche Probleme zu lösen, ist die Verwendung von Gradientenverfahren [17, S. 153]. Ein einfaches Gradientenverfahren zur Minimierung von Zielfunktionen ohne Nebenbedingungen ist "gradient descent"[17, S. 110]. Die Idee dahinter ist es, den

Gradienten, die Richtung des steilsten Anstiegs, der Zielfunktion zu berechnen und sich je nachdem ob die Funktion maximiert oder minimiert werden soll, in die entgegengesetzte oder gleiche Richtung zu bewegen. Dies wird so lange wiederholt, bis ein Minimum gefunden wurde. Eine Herausforderung hierbei ist die Wahl der Schrittweite. In 2 ist beispielhaft ein Gradientenverfahren welches in 32 Schritten terminiert dargestellt. Ein weiteres Problem bei der Verwendung dieser Verfahren ist auch, dass sie je nach Initialisierung oft nur lokale Minima finden [20, S. 9]. Ein Ansatz, um dieses Problem zu umgehen, ist die Verwendung von konvexen Funktionen. Dabei wird die nichtlineare Funktion durch eine konkave Zielfunktion approximiert [20, S. 11]. Dies ist von Vorteil, da konvexe Funktionen nur ein lokales Minimum haben [21, S. 7]. Durch die einfachere Lösbarkeit von konvexen Optimierungsproblemen spielen sie auch in der Literatur eine wichtige Rolle [20, S. 8].

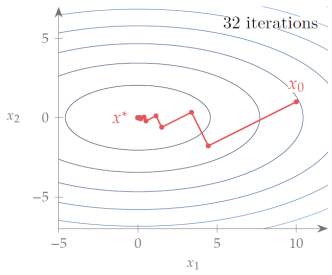


Abbildung 2: Beispiel eines Gradientenverfahren, welchses die Zielfunktion $f(x_1, x_2) = x_1^2 + \beta x_2^2$ minimiert [17, S. 112]

Um auch nichtlineare Probleme mit Nebenbedingungen zu lösen, gibt es verschiedene Ansätze welche auf den oben beschriebenen Verfahren aufbauen. Ein Ansatz hierzu sind beispielsweise Penalty-Methoden. Hierbei wird die Zielfunktion um einen Strafterm $\pi(x)$ erweitert, wenn die Nebenbedingungen verletzt werden. Optimiert wird dann die um den Strafterm erweiterte Zielfunktion $\hat{f}(x) = f(x) + \mu\pi(x)$. [17, S. 175]

3 Recent Approaches To Accelerate Constraint Programs

3.1 Portfolio Ansätze

Die Idee hinter Portfolio Ansätzen ist es, mehrere Solver zu kombinieren, um so die Performance zu steigern.

Ein Beispiel eines Solvers, welcher in vielen SAT-Portfolio-Ansätzen verwendet wird, ist der MiniSat Solver [22]. Dieser basiert im Wesentlichen auf dem Davis-Putnam-Logemann-Loveland-Algorithmus (DPLL) [23], einer speziellen Form des Backtracking-Algorithmus 2.1, der um Unit Propagation erweitert wurde. Unit Propagation beschreibt das Belegen einer Variable, wenn sie in einer Klausel als einzige Variable noch unbelegt ist [5, S.89]. Außerdem verwendet MiniSat weitere Techniken, um die Performance zu

steigern. Dazu zählen zum Beispiel effiziente Datenstrukturen. Beispielsweise wird für jede Variable eine Liste von Klauseln geführt, in denen sie vorkommt. Dadurch können Unit Propagations schneller durchgeführt werden, sobald diese belegt ist, da nur über relevante Klauseln iteriert werden muss. Um die Iteration weiter einzuschränken, wird eine Watched-Literal-Liste geführt, in der für jede Klausel zwei repräsentative unbelegte Literale geführt werden. Dadurch kann schneller erkannt werden, ob die Belegung einer Variable zu einer Unit Propagation führt [22, S. 505]. Des Weiteren wird ein Dynamic Variable Ordering verwendet, bei dem die Literale nach ihrer Aktivität in kürzlichen Konflikten geordnet werden. Bei der Auswahl des nächsten Literals wird dann das aktivste mögliche Literal gewählt, um die Suche auf erfolgversprechende Pfade zu lenken. Diese Art der Suche wird oft als Variable State Independent Decaying Sum (VSIDS) Heuristik bezeichnet [22, 506f].

Es gibt verschiedene Ansätze, wie die Solver dabei kombiniert werden können. Zum einen besteht die Möglichkeit, die Solver statisch festzulegen, als auch dynamisch zu wählen. Weiter besteht die Möglichkeit des Automatischen Parameter-Tunings, bei dem die Parameter der Solver automatisch an das Problem angepasst werden [24, S. 8–11]. Eine weitere Fragestellung ist die Wahl der Solver und wann dieser eingesetzt werden soll. Der am weitesten verbreitete Ansatz ist, einen einzelnen Solver zu Beginn der Laufzeit zu wählen und für das ganze Problem zu verwenden. Es gibt jedoch auch Ansätze, die den Solver zur Laufzeit wechseln oder auch mehrere Solver aus dem Portfolio gleichzeitig verwenden [24, S. 11–14]. Das Hauptproblem besteht jedoch in der richtigen Wahl des Solvers. Während in den Anfängen die Algorithmen nach handgewählten Kriterien ausgewählt wurden, werden heute oft automatische Ansätze verwendet. Dazu zählen Lazy Approaches, wie das Abspeichern von Fallbeispielen oder Nearest-Neighbor Ansätze. Auch wurden schon Klassifikationen, Entscheidungsbäume, Support Vector Machines sowie Neuronale Netze verwendet.

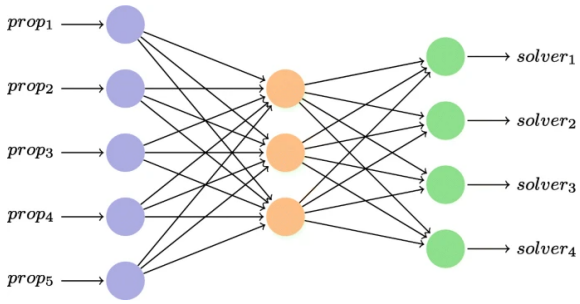


Abbildung 3: Verwendung eines Neuronalen Netzes zur Wahl eines Solvers [25, S. 105]

Bei den Learning Ansätzen besteht jedoch das Problem, dass ein großer Trainingsdatensatz benötigt wird [24, S. 15–16]. Eine weitere Möglichkeit besteht darin, ein Performance-Modell der Solver zu lernen. Dadurch kann einfach ein neuer Solver in das Portfolio aufgenommen werden, da die Modelle für die bestehenden Solver nicht neu gelernt werden müssen [24, S. 18]. Als Features für die Modelle können beispielsweise die Anzahl der Variablen, die Anzahl der Constraints, die Domäne der Variablen oder

Verhältnisse der vorherigen Eigenschaften verwendet werden [24, S. 22]. Ein Beispiel für einen Portfolio Solver ist der Sunny-Cp-Solver, welcher einen lazy k-Nearest-Neighbor zur Auswahl eines Solver-Sets verwendet [16, S. 4].

3.2 Model Based Optimization

Sollte die objektive Funktion nicht bekannt sein und die Auswertung des Blackbox-Modells teuer sein, kann Model Based Optimization verwendet werden, um die Optimierung zu beschleunigen. Hierbei wird zunächst anhand einer kleinen Anzahl von Auswertungen ein Modell der Blackbox erstellt, und dieses Modell wird dann für die Optimierung verwendet. Auf diesem wird solange gearbeitet, bis ein bestimmtes Budget, beispielsweise Schritte oder ein Delta-Wert, erreicht wird. Mit diesem wird dann durch Auswertung der Blackbox das Ergebnis bestimmt, welches entweder verwendet wird, um das Modell zu verfeinern, oder es wird als Endresultat ausgegeben [26, S. 4].

3.3 Automated Parameter Tuning

Ein weiterer Ansatz, um die Performance von Solvern zu steigern, ist das Automated Parameter Tuning. Hierbei werden die Parameter der Solver automatisch an das Problem angepasst:

$$\lambda^* \in \arg\max p(\mathcal{A}_\lambda, \mathcal{D})$$

Hierbei beschreibt λ die Parameter des Algorithmus, $\mathcal{A}$ den Algorithmus und $\mathcal{D}$ die Domäne. λ^* beschreibt die optimalen Parameter für das Problem. Die Funktionsweise des Parameter Tunings ist iterativ. Grundbaustein ist ein Konfigurator, welcher die Parameter des Algorithmus anpasst. Initial werden dem Konfigurator die Parameter und die Domäne übergeben. Der Konfigurator probiert verschiedene Konfigurationen aus und übergibt sie dem Zielalgorithmus. Der Algorithmus wird auf das Problem angewendet und gibt in Form einer Kostenfunktion an, wie gut die Konfiguration war. Der Konfigurator passt iterativ auf Basis der Kostenfunktion die Parameter an [27, S. 31–38].

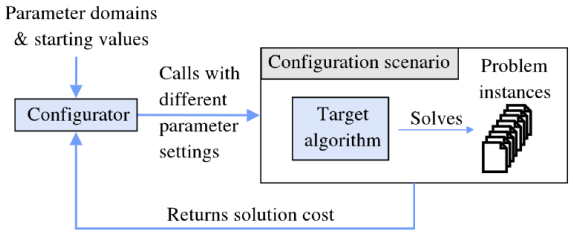


Abbildung 4: Illustration des Parameter-Tuning-Prozesses [27, S. 34]

Die Suche nach den optimalen Parametern kann entweder durch Grid Search oder durch Random Search erfolgen. Bei der Grid Search werden die Parameter systematisch abgedeckt. Bei der Random Search werden die Parameter zufällig ausgewählt. Grid

Search ist oft ineffizient, da eine große Anzahl von unwichtigen Parameterkombinationen getestet werden. Dafür ist es aber unwahrscheinlich, dass ein relevanter Bereich ausgelassen wird. Random Search testet hingegen relevante Parameterkombinationen, jedoch auf zufällige Weise. Dafür arbeitet Grid Search oft schneller, da mehr relevante Parameterkombinationen getestet werden [27, S. 39].

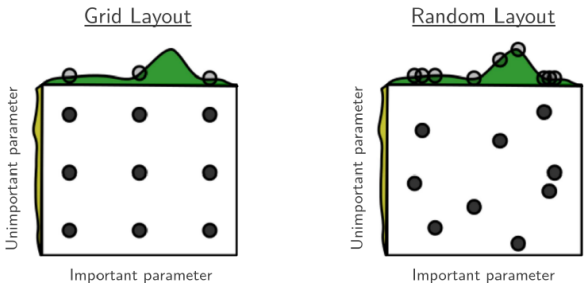


Abbildung 5: Grid Search und Random Search [27, S. 39]

Ein weiterer Ansatz ist Local Search, bei dem ein einzelner Parameter iterativ angepasst wird. Hierbei wird ein einzelner Parameter ausgewählt und verändert. Falls die Veränderung zu einer Verbesserung führt, wird der Parameter behalten, ansonsten verworfen.

3.4 Kombination von CP und SAT

Ein weiterer Ansatz, um CP-Probleme zu beschleunigen, ist die Kombination von CP mit SAT. Hierzu wird die sogenannte Lazy Clause Generation verwendet. Diese Methode wandelt die Constraints eines CP-Problems in SAT-Klauseln um. Diese Klauseln werden nicht im Voraus generiert, sondern erst, wenn sie benötigt werden. Durch die Verwendung von SAT Solvern können die Klauseln schneller gelöst werden. Wenn es bei der Lösung eines SAT-Problems zu einem Konflikt kommt, wird eine Klausel dem Solver hinzugefügt. Durch diese Kombination werden die Stärken von CP (z. B. feinkörnige Domänenreduktion) und SAT (z. B. leistungsstarke Konfliktlösung und Backtracking) optimal genutzt, was zu einem effizienteren und leistungsfähigeren Solver führt [28, S. 5]. Ein Beispiel für einen Solver, der diese Methode verwendet, ist der Chuffed Solver [12].

4 Schluss

Die vorgestellten Ansätze zur Beschleunigung von Constraint Programming (CP) bieten vielfältige Möglichkeiten, die Leistungsfähigkeit von CP-Systemen zu verbessern und damit die Lösung komplexer Probleme effizienter zu gestalten.

Portfolio-Ansätze zeigen, dass die Kombination mehrerer Solver eine vielversprechende Strategie ist, um die Leistung zu steigern. Die dynamische Auswahl oder die Kombination verschiedener Solver je nach Problemstellung kann zu verbesserten Ergebnissen führen. Model Based Optimization und Automated Parameter Tuning bieten weitere Wege, um

die Effizienz von CP zu erhöhen, indem Modelle verwendet oder Parameter automatisch angepasst werden.

Die Kombination von CP und SAT sowie die Nutzung von Lazy Clause Generation stellen ebenfalls vielversprechende Ansätze dar, um die Leistung von CP-Systemen zu steigern. Durch die Integration der Stärken beider Ansätze können komplexe Probleme effizienter gelöst werden.

Es wird jedoch deutlich, dass es keinen universellen Ansatz gibt, um CP-Probleme zu beschleunigen, da die Effektivität der Methoden stark von der spezifischen Problemstellung abhängt. Vielmehr ist ein ganzheitlicher Ansatz erforderlich, der die Auswahl und Kombination verschiedener Techniken je nach Problemstellung und Ressourcen ermöglicht.

Die fortlaufende Forschung und Entwicklung auf diesem Gebiet verspricht, die Leistungsfähigkeit von CP-Systemen weiter zu verbessern und ihre Anwendbarkeit auf eine Vielzahl komplexer realer Probleme zu erweitern.

Es ist anzumerken, dass die vorgestellten Ansätze nur einen Teil des breiten Spektrums der Forschung im Bereich der Beschleunigung von Constraint Programming darstellen. Zukünftige Innovationen könnten auch durch neue Technologien wie Quantum Computing beeinflusst werden.

Insgesamt bleibt die Optimierung von CP-Systemen ein aktives und vielschichtiges Forschungsfeld, das weiterhin großes Potenzial für Innovationen und Fortschritte bietet.

Literatur

[1] *CP 2024*. URL: https://cp2024.a4cp.org/ (besucht am 14. 06. 2024).

[2] Didier Dubois. „Possibility Theory in Constraint Satisfaction Problems: Handling Priority, Preference and Uncertainty“. In: *Applied Intelligence* 6.4 (Okt. 1996), S. 287–309. ISSN: 0924-669X, 1573-7497. DOI: 10.1007/BF00132735. URL: http://link.springer.com/10.1007/BF00132735 (besucht am 14. 06. 2024).

[3] Ridvan Gedik u. a. „Analysis of a Parallel Machine Scheduling Problem with Sequence Dependent Setup Times and Job Availability Intervals“. In: *European Journal of Operational Research* 251.2 (Juni 2016), S. 640–650. ISSN: 03772217. DOI: 10.1016/j.ejor.2015.11.020. URL: https://linkinghub.elsevier.com/retrieve/pii/S0377221715010693 (besucht am 14. 06. 2024).

[4] Liliana Ardissono u. a. „A Framework for the Development of Personalized, Distributed Web-Based Configuration Systems“. In: ().

[5] Francesca Rossi, Peter Van Beek und Toby Walsh, Hrsg. *Handbook of Constraint Programming*. 1st ed. Foundations of Artificial Intelligence. Amsterdam ; Boston: Elsevier, 2006. 955 S. ISBN: 978-0-444-52726-4.

[6] Augustin Delecluse, Pierre Schaus und Pascal Van Hentenryck. „Sequence Variables for Routing Problems“. In: *LIPIcs, Volume 235, CP 2022* 235 (2022), 19:1–19:17. ISSN: 1868-8969. DOI: 10.4230/LIPICS.CP.2022.19. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.CP.2022.19 (besucht am 12. 06. 2024).

[7] Philippe Laborie u. a. „IBM ILOG CP Optimizer for Scheduling“. In: *Constraints* 23.2 (1. Apr. 2018), S. 210–250. ISSN: 1572-9354. DOI: 10.1007/s10601-018-9281-x. URL: https://doi.org/10.1007/s10601-018-9281-x (besucht am 13. 06. 2024).

[8] *IBM ILOG CPLEX Optimization Studio*. URL: https://www.ibm.com/de-de/products/ilog-cplex-optimization-studio (besucht am 13. 06. 2024).

[9] *3.3. Solving Technologies and Solver Backends — The MiniZinc Handbook 2.4.3*. URL: https://docs.minizinc.dev/en/2.4.3/solvers.html (besucht am 13. 06. 2024).

[10] *OR-Tools*. Google for Developers. URL: https://developers.google.com/optimization?hl=de (besucht am 13. 06. 2024).

[11] *GECODE - An Open, Free, Efficient Constraint Solving Toolkit*. URL: https://www.gecode.org/ (besucht am 13. 06. 2024).

[12] *Chuffed/Chuffed*. Chuffed, 21. Mai 2024. URL: https://github.com/chuffed/chuffed (besucht am 13. 06. 2024).

[13] *MiniZinc*. URL: https://www.minizinc.org/ (besucht am 14. 06. 2024).

[14] *Home | MiniZinc Challenge 2024*. URL: https://challenge.minizinc.org/ (besucht am 14. 06. 2024).

[15] Peter Gritzmann. *Grundlagen der Mathematischen Optimierung: Diskrete Strukturen, Komplexitätstheorie, Konvexitätstheorie, Lineare Optimierung, Simplex-Algorithmus, Dualität*. Wiesbaden: Springer Fachmedien Wiesbaden, 2013. ISBN: 978-3-528-07290-2 978-3-8348-2011-2. URL: https://link.springer.com/10.1007/978-3-8348-2011-2 (besucht am 14. 06. 2024).

[16] Roberto Amadini. „Portfolio Approaches in Constraint Programming“. In: *Constraints* 20.4 (Okt. 2015), S. 483–483. ISSN: 1383-7133, 1572-9354. DOI: 10.1007/s10601-015-9215-9. URL: http://link.springer.com/10.1007/s10601-015-9215-9 (besucht am 09. 11. 2023).

[17] Joaquim R. R. A. Martins und Andrew Ning. *Engineering Design Optimization*. 1. Aufl. Cambridge University Press, 18. Nov. 2021. ISBN: 978-1-108-98064-7 978-1-108-83341-7. URL: https://www.cambridge.org/core/product/identifier/9781108980647/type/book (besucht am 15. 06. 2024).

[18] R. J. Dakin. „A Tree-Search Algorithm for Mixed Integer Programming Problems“. In: *The Computer Journal* 8.3 (1. März 1965), S. 250–255. ISSN: 0010-4620, 1460-2067. DOI: 10.1093/comjnl/8.3.250. URL: https://academic.oup.com/comjnl/article-lookup/doi/10.1093/comjnl/8.3.250 (besucht am 15. 06. 2024).

[19] Petra Hofstedt und Armin Wolf. *Einführung in die Constraint-Programmierung: Grundlagen, Methoden, Sprachen, Anwendungen*. eXamen.press. Berlin Heidelberg: Springer, 2007. 388 S. ISBN: 978-3-540-23184-4.

[20] Stephen P. Boyd und Lieven Vandenberghe. *Convex Optimization*. Cambridge, UK ; New York: Cambridge University Press, 2004. 716 S. ISBN: 978-0-521-83378-3.

[21] Jorge Nocedal und Stephen J. Wright. *Numerical Optimization*. 2nd ed. Springer Series in Operations Research. New York: Springer, 2006. 664 S. ISBN: 978-0-387-30303-1.

[22] Niklas Eén und Niklas Sörensson. „An Extensible SAT-solver“. In: *Theory and Applications of Satisfiability Testing*. Hrsg. von Enrico Giunchiglia und Armando Tacchella. Bearb. von Gerhard Goos, Juris Hartmanis und Jan Van Leeuwen. Bd. 2919. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, S. 502–518. ISBN: 978-3-540-20851-8 978-3-540-24605-3. DOI: 10.1007/978-3-540-24605-3_37. URL: http://link.springer.com/10.1007/978-3-540-24605-3_37 (besucht am 27. 07. 2024).

[23] Martin Davis, George Logemann und Donald Loveland. „A Machine Program for Theorem-Proving“. In: *Communications of the ACM* 5.7 (Juli 1962), S. 394–397. ISSN: 0001-0782, 1557-7317. DOI: 10.1145/368273.368557. URL: https://dl.acm.org/doi/10.1145/368273.368557 (besucht am 27. 07. 2024).

[24] Lars Kotthoff. „Algorithm Selection for Combinatorial Search Problems: A Survey“. 30. Okt. 2012. arXiv: 1210.7959 [cs]. URL: http://arxiv.org/abs/1210.7959 (besucht am 15. 06. 2024).

[25] Andrei Popescu u. a. „An Overview of Machine Learning Techniques in Constraint Solving“. In: *Journal of Intelligent Information Systems* 58.1 (Feb. 2022), S. 91–118. ISSN: 0925-9902, 1573-7675. DOI: 10.1007/s10844-021-00666-5. URL: https://link.springer.com/10.1007/s10844-021-00666-5 (besucht am 15. 06. 2024).

[26] Bernd Bischl u. a. *mlrMBO: A Modular Framework for Model-Based Optimization of Expensive Black-Box Functions*. 3. Dez. 2018. arXiv: 1703.03373 [stat]. URL: http://arxiv.org/abs/1703.03373 (besucht am 11. 06. 2024). Vorveröffentlichung.

[27] Lars Kotthoff. „Getting the Best out of Your Constraint Solver: Portfolios and Automated Tuning“. ACP Summer School (Leuven). 11. Juli 2023.

[28] Gerhard Goos u. a. „Lecture Notes in Computer Science“. In: ().

Eidesstattliche Erklärung

Ich versichere, dass ich die vorliegende Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe, und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat.

Alle Ausführungen der Arbeit, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Bei der Erstellung dieses Dokuments wurde künstliche Intelligenz (KI)-basierte Software zur Generierung von Inhalten verwendet.

Augsburg, —

(—)